Efficient mining Top-k Regular-frequent itemset using Compressed Tidsets

Komate Amphawan[1,2,3], Philippe Lenca[2,3], and Athasit Surarerks[1]

[1] Chulalongkorn University, ELITE laboratory, 10330 Bangkok, Thailand
komate@live.com, Athasit.S@chula.ac.th
[2] Institut Telecom, Telecom Bretagne, UMR CNRS 3192 Lab-STICC, France
philippe.lenca@telecom-bretagne.eu
[3] Université européenne de Bretagne

Abstract. Association rule discovery based on support-confidence framework is an important task in data mining. However, the occurrence frequency (support) of a pattern (itemset) may not be a sufficient criterion for discovering interesting patterns. Temporal regularity, which can be a trace of behavior, with frequency behavior can be revealed as an important key in several applications. A pattern can be regarded as a regular pattern if it occurs regularly in a user-given period. In this paper, we consider the problem of mining top-k regular-frequent itemsets from transactional databases without support threshold. A new concise representation, called *compressed transaction-ids set (compressed tidset)*, and a single pass algorithm, called *TR-CT (Top-k Regular frequent itemset mining based on Compressed Tidsets)*, are proposed to maintain occurrence information of patterns and discover k regular itemsets with highest supports, respectively. Experimental results show that the use of the compressed tidset representation achieves highly efficiency in terms of execution time and memory consumption, especially on dense datasets.

1 Introduction

The significance of regular-frequent itemsets with temporal regularity can be revealed in a wide range of applications. Regularity is a trace of behavior and as pointed out by [1], behaviors can be seen everywhere in business and social life. For example in commercial web site analysis, one can be interested to detect such frequent regular access sequences in order to assist in browsing the Web pages and to reduce the access time [2, 3]. In a marketing point of view, managers will be interested in frequent regular behavior of customers to develop long-term relationships but also to detect changes in customer behavior [4].

Tanbeer et al. [5] proposed to consider the occurrence behavior of patterns *i.e.* whether they occurs regularly, irregularly or mostly in specific time period of a transactional database. A pattern is said regular-frequent if it is frequent (as defined in [6] thanks to the support measure) and if it appears regularly (thanks to a measure of regularity/periodicity which considers the maximum compressed at which the pattern occurs).

To discover a set of regular-frequent itemsets, the authors proposed a highly compact tree structure, named *PF-tree (Periodic Frequent patterns tree)*, to maintain the database content, and a pattern growth-based algorithm to mine a complete set of regular-frequent itemsets with the user-given support and regularity thresholds. This approach has been extended on incremental transactional databases [7], on data stream [8] and mining periodic-frequent patterns consisting of both frequent and rare items [9].

However, it is well-known that support-based approaches tend to produce a huge number of patterns and that it is not easy for the end-users to define a suitable support threshold. Thus, top-k patterns mining framework, which allows the user to control the number of patterns (k) to be mined (which is easy to specify) without support threshold, is an interesting approach [10].

In [11] we thus proposed to mine the top-k regular-frequent patterns and the algorithm MTKPP (Mining Top-K Periodic-frequent Patterns). MTKPP discovers the set of k regular patterns with highest support. It scans the database once to collects the set of transaction-ids where each item occurs in order to calculate their supports and regularities. Then, it requires an intersection operation on the transaction-ids set to calculate the support and the regularity of each itemset. This operation is the most memory and time consuming process.

In this paper, we thus propose a compressed tidset representation to maintain the occurrence information of itemsets to be mined. Indeed, compressed representation for intersection operation have shown their efficient like in Diffsets [12] and bit vector [13]. Moreover, an efficient single-pass algorithm, called *TR-CT (Top-k Regular-frequent itemsets mining based on Compressed Tidsets)* is proposed. The experimental results show that the proposed TR-CT algorithm achieves less memory usage and execution time, especially on dense datasets for whose the compressed tidset representation is very efficient.

The problem of top-k regular-frequent itemsets mining is presented in Section 2. The compressed tidset representation and the proposed algorithm are described in Section 3. In Section 4, we compare the performance of TR-CT algorithm with MTKPP. Finally, we conclude in Section 5.

2 Top-k Regular-frequent itemsets mining

In this section, we introduce the basic definitions used to mine regular-frequent itemsets [5] and top-k regular-frequent itemsets [11].

Let $I = \{i_1, \ldots, i_n\}$ be a set of items. A set $X = \{i_{j_1}, \ldots, i_{j_l}\} \subseteq I$ is called an *itemset* or an *l-itemset (an itemset of size l)*. A transactional database $TDB = \{t_1, t_2, \ldots, t_m\}$ is a set of transactions in which each transaction $t_q = (q, Y)$ is a tuple containing a unique transaction identifier q (tid in the latter) and an itemset Y. If $X \subseteq Y$, it is said that t_q contains X (or X occurs in t_q) and is denoted as t_q^X. Therefore, $T^X = \{t_p^X, \ldots, t_q^X\}$, where $1 \leq p \leq q \leq |TDB|$, is the set of all ordered tids (called *tidset*) where X occurs. The support of an itemset X, denoted as $s^X = |T^X|$, is the number of tids (transactions) in TDB where X appears.

Definition 1 (Regularity of an itemset X). *Let t_p^X and t_q^X be two consecutive tids in T^X, i.e. where $p < q$ and there is no transaction t_r, $p < r < q$, such that t_r contains X (note that p, q and r are indices). Then, $rtt_q^X = t_q^X - t_p^X$ represents the number of tids (transactions) not containing X between the two consecutive transactions t_p^X and t_q^X.*

To find the exact regularity of X, the first and the last regularities are also calculated : (i) *the first regularity of $X(fr^X)$ is the number of tids not containing X before it first occurs (i.e. $fr^X = t_1^X$), and* (ii) *the last regularity (lr^X) is the number of tids not containing X from the last occurring of X to the last tids of database (i.e. $lr^X = |TDB| - t_{|T^X|}^X$).*

Thus, the regularity of X is defined as $r^X = max(fr^X, rtt_2^X, rtt_3^X, \ldots, rtt_{|T^X|}^X, lr^X)$ which is the maximum number of tids that X does not appear in database.

Definition 2 (Top-k regular-frequent itemsets). *Let us sort itemsets by descending support values, let S_k be the support of the k^{th} itemset in the sorted list. The top-k regular-frequent itemsets are the set of first k itemsets having highest supports (their supports are greater or equal to S_k and their regularity are no greater than the user-given regularity threshold σ_r).*

Therefore, the top-k regular-frequent itemsets mining problem is to discover k regular-frequent itemsets with highest support from *TDB* with two user-given parameters: the number k of expected outputs and the regularity threshold (σ_s).

3 TR-CT: Top-k Regular-frequent itemsets mining based on Compressed Tidsets

We now introduce an efficient algorithm, called *TR-CT*, to mine the top-k regular-frequent itemset from a transactional database. It uses a concise representation, called *compressed transaction-ids set (compressed tidset)* to maintain the occurrence information of each itemset. It also uses an efficient data structure, named *top-k list* (as proposed in [11]) to maintain essential information about the top-k regular-frequent itemsets.

3.1 Compressed tidset representation

The compressed tidset representation is a concise representation used to store the occurrence information (tidset: a set of tids that each itemset appears) of the top-k regular-frequent itemsets during mining process. The main concept of the compressed tidset representation is to wrap up two or more consecutive continuous tids by maintaining only the first (with one positive integer) and the last tids (with one negative integer) of that group of tids. TR-CT can thus reduce time to compute support and regularity, and also memory to store occurrence information. In particular this representation is appropriate for dense datasets.

Definition 3 (Compressed tidset of an itemset X). *Let $T^X = \{t_p^X, t_{p+1}^X, \ldots, t_q^X\}$ be the set of tids that itemset X occurs in transactions where $p < q$ and there are some consecutive tids $\{t_u^X, t_{u+1}^X, \ldots, t_v^X\}$ that are continuous between t_p^X and t_q^X **(where** $p \leq u$ **and** $q \geq v$**)**. Thus, we define the compressed tidset of itemset X as:*

$$CT^X = \{t_p^X, t_{p+1}^X, \ldots, t_u^X, (t_u^X - t_v^X), t_{v+1}^X, \ldots, T_q^X\}$$

This representation is efficient as soon as there are three consecutive continuous transaction-ids in the tidsets. In the worst case, the compressed representation of a tidset is equal of the size of the tidset.

Table 1. A transactional database as a running example of TR-CT

tid	items
1	*a b c d f*
2	*a b d e*
3	*a c d*
4	*a b*
5	*b c e f*
6	*a d e*
7	*a b c d e*
8	*a b d*
9	*a c d f*
10	*a b e*
11	*a b c d*
12	*a d f*

From the TDB on the left side we have $T^a = \{t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}\}$ which is composed of two groups of consecutive continuous transactions. Thus, the compressed tidset of item a is $CT^a = \{1, -3, 6, -6\}$. For example, the first compressed tids $(1, -3)$ represents $\{t_1, t_2, t_3, t_4\}$ whereas $(6, -6)$ represents the last seven consecutive continuous tids. For the item a, the use of compressed tidset representation is efficient. It can reduce seven tids to be maintained comparing with the normal tidset representation. For items b and c, the sets of transactions that they occur are $T^b = \{t_1, t_2, t_4, t_5, t_7, t_8, t_{10}, t_{11}\}$ and $T^c = \{t_1, t_3, t_5, t_7, t_9, t_{11}\}$, respectively. Therefore, the compressed tidsets of the items b and c are $CT^b = \{1, -1, 4, -1, 7, -1, 10, -1\}$ and $CT^c = \{1, 3, 5, 7, 9, 11\}$ which are the examples of the worst cases of the compressed tidset representation.

With this representation a tidset of any itemset may contain some negative tids and the original Definition 1 is not suitable. Thus, we propose a new way to calculate the regularity of any itemset from the compressed tidset representation.

Definition 4 (Regularity of an itemset X from compressed tidset). *Let t_p^X and t_q^X be two consecutive tids in compressed tidset CT^X, i.e. where $p < q$ and there is no transaction t_r, $p < r < q$, such that t_r contains X (note that p, q and r are indices). Then, we denote rtt_q^X as the number of tids (transactions) between t_p^X and t_q^X that do not contain X. Obviously, rtt_1^X is t_1^X. Last, to find the exact regularity of X, we have to calculate the number of tids between the last tid of CT^X and the last tid of the database. This leads to the following cases:*

$$rtt_q^X = \begin{cases} t_q^X & \text{if } q = 1 \\ t_q^X - t_p^X & \text{if } t_p^X \text{ and } t_q^X > 0, 2 \leq q \leq |CT^X| \\ 1 & \text{if } t_p^X > 0 \text{ and } t_q^X < 0, 2 \leq q \leq |CT^X| \\ t_q^X + (t_p^X - t_{p-1}^X) & \text{if } t_p^X < 0 \text{ and } t_q^X > 0, 2 \leq q \leq |CT^X| \\ |TDB| - t_{|CT^X|}^X & \text{if } t_{|CT^X|}^X > 0, \text{ (i.e. } q = |CT^X| + 1\text{)} \\ |TDB| + (t_{|CT^X|}^X - t_{|CT^X|-1}^X) & \text{if } t_{|CT^X|}^X < 0, \text{ (i.e. } q = |CT^X| + 1\text{)} \end{cases}$$

Finally, we define the regularity of X *as* $r^X = max(rtt_1^X, rtt_2^X, \ldots, rtt_{m+1}^X)$.

For example, consider the compressed tidset $CT^a = \{1, -3, 6, -6\}$ of item a. The set of regularities between each pair of two consecutive tids is $\{1, 1, 6 + (-3 - 1), 1, 12 - (-6 - 6)\} = \{1,1,2,1,0\}$ and the regularity of item a is 2.

3.2 Top-k list structure

As in [11], TR-CT is based on the use of a top-k list, which is an ordinary linked-list, to maintain the top-k regular-frequent itemsets. A hash table is also used with the top-k list in order to quickly access each entry in the top-k list. As shown in Fig. 1, each entry in a top-k list consists of 4 fields: (i) an item or itemset name (I), (ii) a total support (s^I), (iii) a regularity (r^I) and ($iiii$) an compressed tidset where I occurs (CT^I). For example, an item a has a support of 11, a regularity of 2 and its compressed tidset is $CT^a = \{1, -3, 6, -6\}$ (Fig. 1(d)).

3.3 TR-CT algorithm description

The TR-CT algorithm consists of two steps: (i) Top-k list initialization: scan database once to obtain and collect the all regular items (with highest support) into the top-k list; (ii) Top-k mining: use the best-first search strategy to cut down the search space, merge each pair of entries in the top-k list and then intersect their compressed tidsets in order to calculate the support and the regularity of a new generated regular itemset.

Top-k initialization. To create the top-k list, TR-CT scans the database once transaction per transaction. Each item of the current transaction is then considered. Thanks to the help of the hash table we know quickly if the current item is already in the top-k list or not. In the first case we just have to update its support, regularity and compressed tidset. If it is its first occurrence then a new entry is created and we initialize its support, regularity and compressed tidset.

To update the compressed tidset CT^X of an itemset X, TR-CT has to compare the last tid (t_i) of CT^X with the new coming tid (t_j). Thanks to the compressed representation (see Definition 3) it simply consists into the following cases:

- if $t_i < 0$, *i.e.* there are former consecutive continuous tids occur with the exact tid of t_i. TR-CT calculates the exact tid of $t_i < 0$ (*i.e.* $t_{i-1} - t_i$) and compares it with t_j to check whether they are continuous. If they are consecutive continuous tids (*i.e.* $t_j - t_{i-1} + t_i = 1$), TR-CT has to extend the compressed tidset CT^X (it consists only of adding -1 to t_i). Otherwise, TR-CT adds t_j after t_i in CT^X.
- if $t_i > 0$, *i.e.* there is no former consecutive continuous tid occurs with t_i. TR-CT compared t_i with t_j to check whether they are continuous. If they are consecutive continuous tids (i.e. $t_j - t_i = 1$), TR-CT creates a new tid in CT^X (it consists of adding -1 after t_i in CT^X). Otherwise, TR-CT adds t_j after t_i in CT^X.

After scanning all transactions, the top-k list is trimmed by removing all the entries (items) with regularity greater than the regularity threshold σ_r, and the remaining entries are sorted in descending order of support. Lastly, TR-CT removes the entries after the k^{th} entry in the top-k list.

Top-k mining. A best-first search strategy (from the most frequent itemsets to the least frequent itemsets) is adopted to quickly generate the regular itemsets with highest supports from the top-k list.

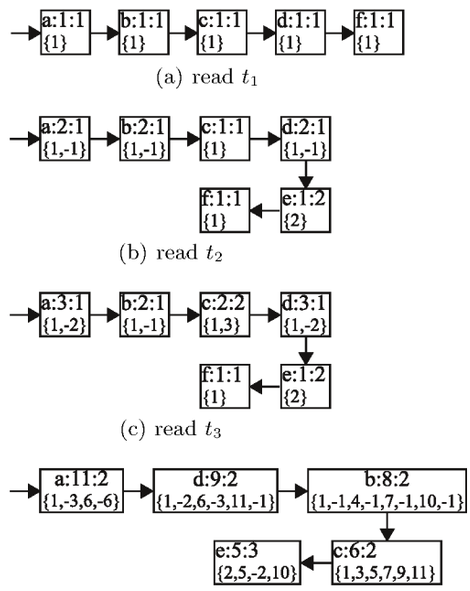
Two candidates X and Y in the top-k list are merged if both itemsets have the same prefix (*i.e.* each item from both itemsets is the same, except the last item). This way of doing will help our algorithm to avoid the repetition of generating larger itemsets and can help to prune the search space. After that, the compressed tidsets of the two elements are sequentially intersected in order to calculate the support, the regularity and the compressed tidset of the new generated itemset. To sequentially intersect compressed tidsets CT^X and CT^Y of X and Y, one has to consider four cases when comparing tids t_i^X and t_j^Y in order to construct CT^{XY} (see Definition 3):

(*1*) if $t_i^X = t_j^Y > 0$ add t_i^X at the end of CT^{XY}
(*2*) if $t_i^X > 0, t_j^Y < 0, t_i^X \leq t_{j-1}^Y - t_j^Y$, add t_i^X at the end of CT^{XY}
(*3*) if $t_i^X < 0, t_j^Y > 0, t_j^Y \leq t_{i-1}^X - t_i^X$, add t_j^Y at the end of CT^{XY}
(*4*) if $t_i^X, t_j^X < 0$, add $t^{XY}_{|CT^{XY}|} - (t_{i-1}^X - t_i^X)$ at the end of CT^{XY} if $t_{i-1}^X - t_i^X < t_{j-1}^Y - t_j^Y$ otherwise add $t^{XY}_{|CT^{XY}|} - (t_{j-1}^Y - t_j^Y)$ at the end of CT^{XY}

From CT^{XY} we can easily compute the support s^{XY} and regularity r^{XY} of XY (see definition 4). TR-CT then removes the k^{th} entry and inserts itemset XY into the top-k list if s^{XY} is greater than the support of the k^{th} itemset in the top-k list and if r^{XY} is not greater than the regularity threshold σ_r.

3.4 An example

Consider the TDB of Table 1, a regularity threshold σ_r of 4 and the number of desired results k of 5.



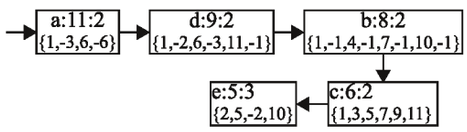
After scanning the first transaction ($t_1 = \{a, b, c, d, f\}$), the entries for items a, b, c, d and f are created and their supports, regularities and compressed tidsets are initialized as $(1 : 1 : \{1\})$ (see Fig. 1(a)). With the second transaction ($t_2 = \{a, b, d, e\}$), TR-CT adds -1 at the end of the compressed tidsets of a, b and d, since these items occur in two consecutive continuous transactions. Then, the entry for item e is created and initialized (Fig. 1(b)). For the third transaction ($t_3 = \{a, c, d\}$), as shown in Fig. 1(c), the last tids of item a and d are changed to -2 (they occur in three consecutive continuous transactions t_1, t_2 and t_3) and the compressed tidset of item c is updated by adding t_3 as the last tid. After scanning all the transactions, the top-k list is sorted by support descending order and item f is removed (Fig. 1(d)).

Fig. 1. Top-k list initialization

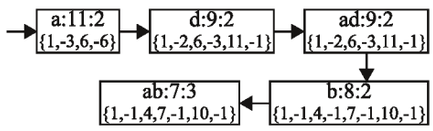
In the mining process, item d is firstly merged with the former item a. The compressed tidsets CT^a and CT^d are sequentially intersected to calculate the support $s^{ad} = 9$, the regularity $r^{ad} = 3$ and to collect the compressed tidset $CT^{ad} = \{1, -2, 6, -3, 11, -1\}$ of itemset ad. Since the support s^{ad} is greater than $s^e = 5$ and the regularity r^{ad} is less than $\sigma_r = 4$, the item e is removed and ad is inserted into the top-k list as shown in Fig. 2(a). Next, the third itemset *i.e.* itemset ad is compared to the former itemsets a and b. These itemsets do not share the same prefix and thus are not merged. TR-CT then considers item b which is merged with a and d ($s^{ab} = 7$, $r^{ab} = 3$, $CT^{ab} = \{1, -2, 7, -1, 10, -1\}$; $s^{bd} = 5$, $r^{bd} = 5$, $CT^{bd} = \{1, -1, 7, -1, 11\}$). The itemset ab is thus added to the list and itemset c is removed and the itemset bd is eliminated. Lastly, the itemsets ab and ad are considered and we finally obtain the top-k regular-frequent itemsets as shown in Fig. 2(b).

4 Performance evaluation

To validate the effectiveness of our proposed TR-CT algorithm, several experiments were conducted to compare the performance of TR-CT with the MTKPP algorithm[11] which is the first algorithm to mine top-k regular-frequent itemsets. All experiments were performed on an Intel®Xeon 2.33 GHz with 4 GB



(a) top-k list when merging item a with item d



(b) final top-k list

Fig. 2. Top-k during mining process

main memory, running Linux platform. Programs are in C. To measure the performance of the both algorithms, we focus on processing time (included top-k list construction and mining processes) and memory consumption (*i.e.* the maximum memory usage of the top-k list during mining process).

4.1 Test environment and datasets

The experiments were performed on several real datasets from the UCI Machine Learning Repository [14]. Table 2 shows some statistical information about the datasets used for experimental analysis. Accidents, connect, and pumsb are dense datasets (with long frequent itemsets) whereas retail is a sparse dataset (with short itemsets). These datasets are used to evaluate the computational performance of our algorithm. The regular and frequent patterns may have no sense (in particular with connect). We are here only interested in the efficiency evaluation (for k between 0 and 10000 and for σ_r between 0.5 and 10% of the total number of transactions in database).

Table 2. Database characteristics

Database	#items	Avg.length	#Transactions	Type
accidents	468	338	340,183	dense
connect	129	43	67,557	dense
pumsb	2,113	74	49,046	dense
retail	16,469	10.3	88,162	sparse

4.2 Execution time

Figures 3, 4, and 5 give the processing time of dense datasets which are accidents, connect, and pumsb, respectively. From these figures, we can see that the proposed TR-CT algorithm runs faster than MKTPP algorithm using normal tids set under various value of k and regularity threshold σ_r. Since the characteristic of dense datasets, TR-CT can take the advantage of the compressed tidset representation which groups consecutive continuous tids together. Meanwhile, the execution time on sparse dataset retail is shown in Figure 6. Note that the performance of TR-CT is similar with MTKPP as with sparse dataset TR-CT can only take the advantage of grouping very few consecutive continuous tids.

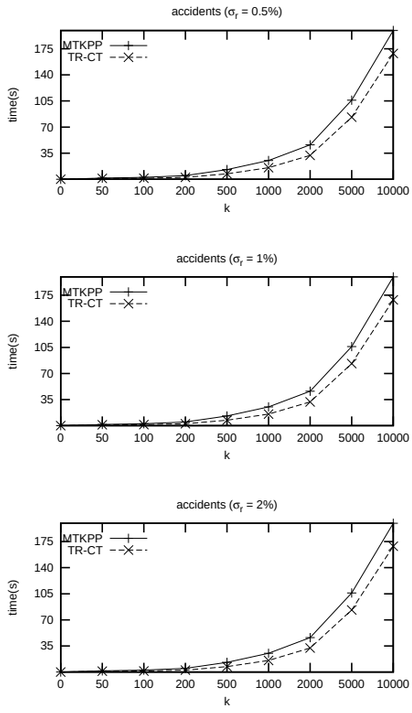


Fig. 3. Performance on accidents

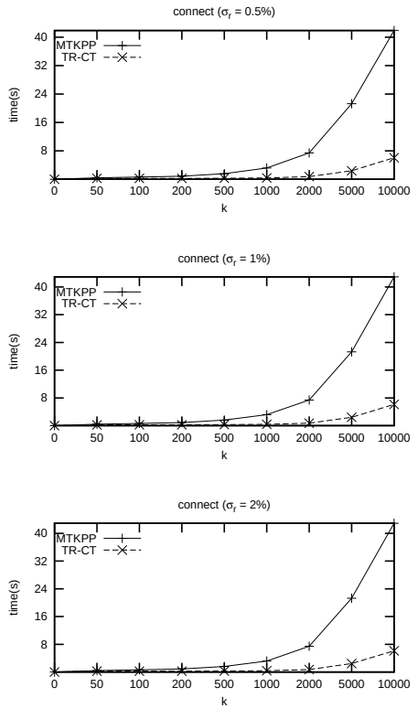


Fig. 4. Performance on Connect

4.3 Space usage

Based on the use of top-k list and compressed tidset representation, the memory usage and the number of maintained tids during mining process are examined. To

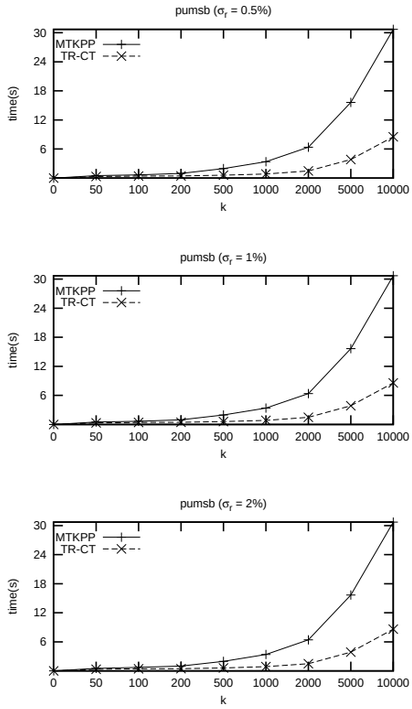


Fig. 5. Performance on Pumsb

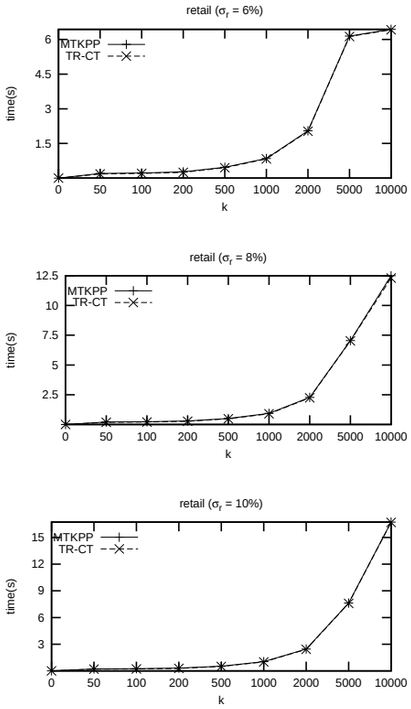


Fig. 6. Performance on Retail

evaluate the space usage, the regularity threshold σ_r is set to be the highest value (used in previous subsection) for each dataset. The first experiment compare the memory consumption of TR-CT and MTKPP algorithm. As shown in Fig. 7, TR-CT uses less memory than that of MTKPP on dense datasets (*i.e. accidents, connect* and *pumsb*) whereas the memory consumption of TR-CT is quite similar as MTKPP on sparse database *retail*. In some cases, the use of the compressed tidset representation may generate more concise tidsets than the original tidsets (used in MTKPP) since the former maintains only the first and last tids of the two or more consecutive continuous tids by using only one positive and one negative integer, respectively. That is why TR-CT has a good performance especially on dense datasets.

In the second experiment, the number of maintained tids is considered (see Fig. 8). The use of the compressed tidset representation may generate more concise tidsets than the original tidsets (used in MTKPP) since the former maintains only the first and last tids of the two or more consecutive continuous tids by using only one positive and one negative integer, respectively. The numbers of maintained tids between the two representations (algorithms) are shown in Fig. 8. It

is observed from the figure that the TR-CT maintained nearly the same number of tids as the MTKPP when dataset are sparse. Meanwhile, TR-CT significantly reduces the number of tids on dense datasets.

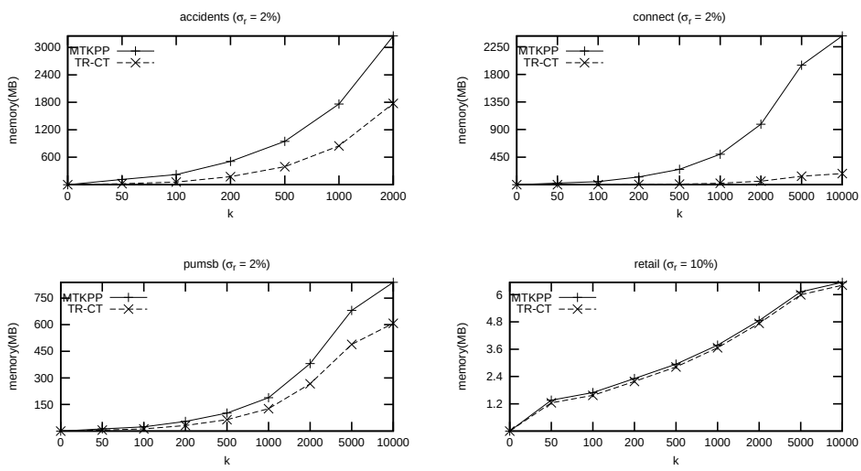


Fig. 7. Memory consumption of TR-CT

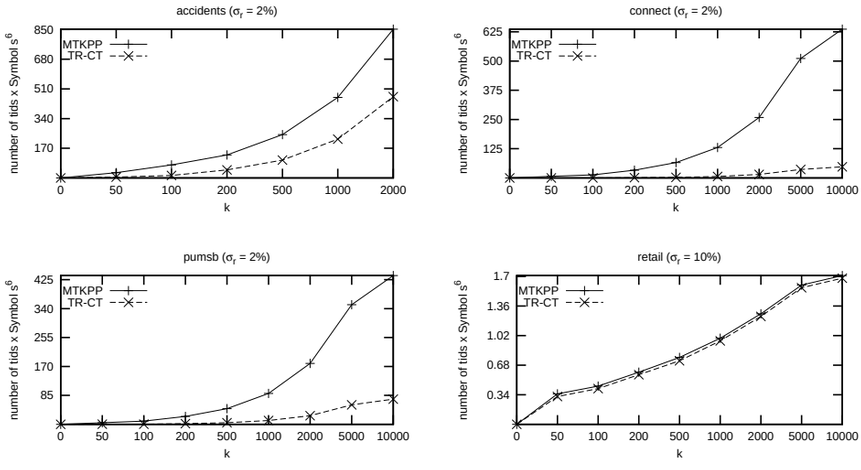


Fig. 8. Number of maintained transaction-ids

5 Conclusion

In this paper, we have studied the problem of mining top-k regular-frequent itemsets mining without support threshold. We propose a new algorithm called TR-CT (Top-k Regular-frequent itemset mining based on Compressed Tidsets) based on a *compressed tidset* representation. By using this representation, a set of tids that each itemset occurs consecutively continuous is transformed and compressed into two tids by using only one positive and negative integer. Then, the top-k regular-frequent itemsets are found by intersection compressed tidsets along the order of top-k list.

Our performance studies on both sparse and dense datasets show that the proposed algorithm achieves high performance, delivers competitive performance, and outperforms MTKPP algorithm. TR-CT is clearly superior to MTKPP on both the small and large values of k when the datasets are dense.

References

1. Cao, L.: In-depth behavior understanding and use: The behavior informatics approach. Inf. Sci. **180**(17) (2010) 3067–3085
2. Shyu, M.L., Haruechaiyasak, C., Chen, S.C., Zhao, N.: Collaborative filtering by mining association rules from user access sequences. In: Int. Workshop on Challenges in Web Information Retrieval and Integration, IEEE Computer Society (2005) 128–135
3. Zhou, B., Hui, S.C., Chang, K.: Enhancing mobile web access using intelligent recommendations. IEEE Intelligent Systems **21**(1) (2006) 28–34
4. Chen, M.C., Chiu, A.L., Chang, H.H.: Mining changes in customer behavior in retail marketing. Expert Syst. Appl. **28**(4) (2005) 773–781
5. Tanbeer, S.K., Ahmed, C.F., Jeong, B.S., Lee, Y.K.: Discovering periodic-frequent patterns in transactional databases. In: PAKDD. Volume 5476 of LNCS., Springer (2009) 242–253
6. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB. (1994) 487–499
7. Tanbeer, S.K., Ahmed, C.F., Jeong, B.S.: Mining regular patterns in incremental transactional databases. In: Int. Asia-Pacific Web Conference, IEEE Computer Society (2010) 375–377
8. Tanbeer, S.K., Ahmed, C.F., Jeong, B.S.: Mining regular patterns in data streams. In: DASFAA. Volume 5981 of LNCS., Springer (2010) 399–413
9. Kiran, R.U., Reddy, P.K.: Towards efficient mining of periodic-frequent patterns in transactional databases. In: DEXA. Volume 6262 of LNCS. (2010) 194–208
10. Han, J., Wang, J., Lu, Y., Tzvetkov, P.: Mining top-k frequent closed patterns without minimum support. In: IEEE ICDM. (2002) 211–218
11. Amphawan, K., Lenca, P., Surarerks, A.: Mining top-k periodic-frequent patterns without support threshold. In: IAIT. Volume 55 of CCIS., Springer (2009) 18–29
12. Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. In: ACM SIGKDD KDDInternational Conference. (2003) 326–335
13. Shenoy, P., Haritsa, J.R., Sudarshan, S., Bhalotia, G., Bawa, M., Shah, D.: Turbo-charging vertical mining of large databases. SIGMOD Rec. **29**(2) (2000) 22–33
14. Asuncion, A., Newman, D.: UCI machine learning repository (2007)